VKT: Verified Knowledge Transfer

for Coding Agents

A Stack Overflow for agents, built on reusable procedures and local verification.

Aymeric Rabot · Julien Brissonneau

Pascal

ABSTRACT

We study *Verified Knowledge Transfer* (VKT), a method developed at Pascal for reusing coding procedures with applicability conditions and verification evidence, without updating model weights. In five runs per arm, relevant records reduced median effort on four selected API-change tasks. On the primary task, median turns fell from 19 to 10 and runtime from 240 to 111 seconds; a second model showed the same direction. Benefits on whole-feature tasks were inconsistent, and one control incurred additional work. The results suggest that shared procedural knowledge can reduce repeated discovery when an applicable solution is concise and costly to recover. Small samples and design limitations leave its general benefit in everyday coding work unresolved.

PRIMARY TASK · VITEST BENCHMARK · BASELINE → VKT RECORD

19 → 10	893k → 383k	240 → 111 s
Median agent turns	Median counted tokens	Median agent runtime

Five runs per arm; all ten passed. Descriptive medians. Tokens include cache reads and writes; runtime excludes the final hidden-test run.

01 From solved problems to shared knowledge

A coding agent can discover a changed API, diagnose an unfamiliar failure, and verify a working fix. Unless that experience is retained and made available, a later session may repeat the same investigation. A shared collection of solutions offers a practical analogy to Stack Overflow: a problem becomes searchable knowledge for future work.

Inspired by the reusable-library perspective of Ellis et al.'s *DreamCoder*, we examine this loop at the level of coding procedures.[1] Related work includes DeepMind's retrieval-enhanced language models, *Voyager*'s executable skill library, and Google Research's *ReasoningBank*.[2–4] *Agent KB* is a close precedent for sharing experience across agent frameworks, while *Agent Workflow Memory* studies the reuse of induced routines.[5,6]

VKT names the method evaluated here: retain a solution procedure and its evidence, retrieve it for a compatible problem, and let the receiving agent adapt and check it. Our focus is when this process reduces effort, particularly on selected dependency changes that the study classified as newer than the model's knowledge.

FIGURE 1 · THE TRANSFER MECHANISM MODEL WEIGHTS STAY FIXED

A procedure is the unit of reuse. The diagram describes the intended workflow, not a measured network effect. The evaluated records include material derived from documentation and from the task fixtures; the study does not establish autonomous transfer between independent teams.

02 A method for reusing procedural knowledge

A record contains an observable problem signature, environment constraints, a proposed procedure, pitfalls, and verification evidence. Retrieval considers the problem and relevant stack information. The receiving agent checks applicability, derives a solution in the current codebase, and runs an appropriate local check. Successful verification in the originating environment supplies evidence for reuse; it does not guarantee correctness elsewhere.

This mechanism operates within the agent's problem-solving process. It does not establish a new task-decomposition algorithm or change the model's token-generation speed. Its potential saving is the work an agent would otherwise spend discovering, testing, and correcting a solution. Exact replay of stored files is a separate, easier case.

> **Research question.** When does a compact record of a previously checked solution reduce the effort required to solve a later coding task?

Experimental design

The main study ran on 7–8 September 2026 under a preregistered protocol with logged extensions. It contains four selected API-change tasks, two known-knowledge controls, and three whole-feature tasks. Each task–condition group has five repeated runs; 125 runs enter the final comparisons after ten superseded runs are excluded. Comparisons use the same model, task prompt, and repository fixture within each base comparison, with or without the memory plugin. All baseline runs preceded the corresponding memory runs.

TASK COVERAGE 9 tasks; 4 selected API changes, 2 knowledge controls, 3 whole-feature builds.	COMPLETED RUNS 135, including 10 replacement runs; the superseded originals are excluded from the final comparisons.
PRIMARY COMPARISON Vitest 5 benchmark, using a record classified as cross-repository in the study.	ADDITIONAL ARMS Wrong record, type definitions, documentation retrieval, official prose, and a second model.

Two API-change records were authored for the evaluation fixture; two were classified as cross-repository. Tasks, records, and verifiers were selected or written by the same team. These experiments therefore test the mechanism on selected cases, rather than estimating its prevalence across coding work.

Outcomes are assistant turns, total counted tokens, agent runtime, and hidden-test success. Tokens include input, output, cache reads, cache writes, and the injected record; most counted tokens are cache reads. Runtime ends when the agent exits, before the final hidden tests. Neither measure is a direct estimate of compute or monetary cost.

Reported tests are exact, two-sided Mann–Whitney permutation tests, with Fisher's exact test for pass rates. A combined test aggregates the four API-change turn comparisons. Multiplicity corrections were added as sensitivity analyses; the protocol did not prespecify a correction family. Full numerical summaries and experimental qualifications appear in the appendix.

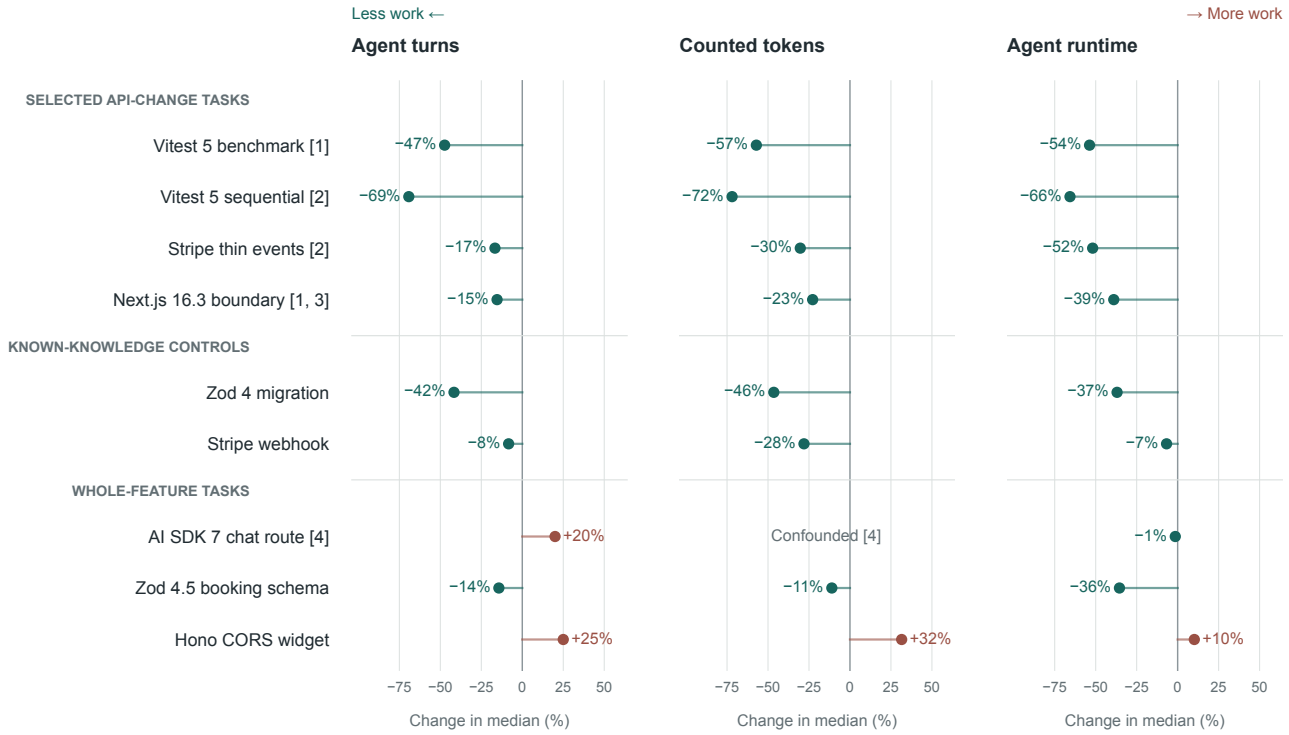
03 Where prior experience reduced effort

All four selected API-change tasks have lower median turns, counted tokens, and runtime with a relevant record. On the primary task, turns decrease by 47%, counted tokens by 57%, and runtime by 54%. Including the final hidden tests, median elapsed time was 249 seconds for baseline and 118 seconds with memory. These percentages compare medians from five runs in each arm. All runs passed on three of the four tasks.

The exception is the Next.js boundary task, whose replacement runs passed in four of five memory runs and none of five baseline runs. Its runtime comparison therefore includes unsuccessful attempts and must not be interpreted as time saved at equal success. Both arms were rerun after a defective verifier was corrected.

FIGURE 2 · ALL TASK COMPARISONS

VKT RECORD RELATIVE TO BASELINE



Benefits depend on the task. Points show percentage changes in reported medians; negative values indicate less work. These are not confidence intervals or paired-run estimates. Each arm has five runs. [1] Cross-repository record, as classified in the study. [2] Record authored for the fixture. [3] Next.js uses replacement runs and has unequal pass rates (0/5 baseline; 4/5 memory). [4] AI SDK token comparisons are omitted because an unrelated skill inflated the arms unevenly. The exact values are in Appendix A.

The combined API-change test for turns gives $p = 0.0030$ under the reported independence assumption. No individual comparison survives Holm correction across the twelve task–outcome tests in that family; the smallest adjusted p is 0.095. The consistent direction is evidence for the mechanism, but these data do not establish a general speedup.

What the record adds beyond an excerpt

On the primary task, the correct record produced lower median turns than the wrong record, a similar-length excerpt of type definitions, a documentation tool, and injected official prose. The controls distinguish different information sources, but they do not isolate the contribution of the error signature, procedure, and verification command individually.

FIGURE 3 · PRIMARY-TASK CONTROLS

FABLE 5.1 · 5 RUNS PER ARM

Medians and full observed ranges. Lines span the minimum and maximum; dots mark medians. All runs in these arms passed. † Documentation-tool and official-prose arms ran in a later window, after two dependencies and three corpus records had been added. Their comparisons carry that environment change. The record's runtime difference from baseline has unadjusted p = 0.0556.

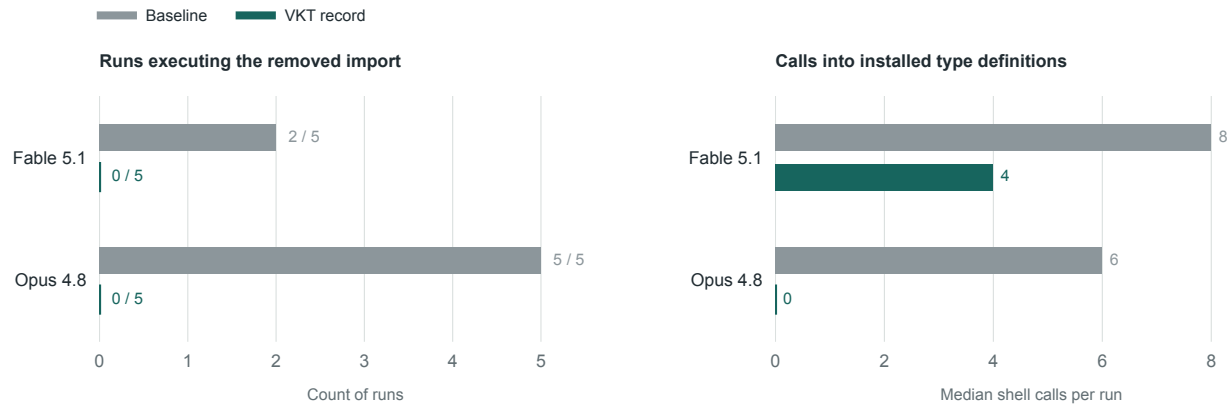
The wrong record did not demonstrate harm relative to baseline. Its false API claim was recoverable by inspecting the installed package; all five runs passed, and one was slower than the baseline median. This tests a readily refutable error, leaving silent incompatibilities untested. The documentation controls likewise do not establish that documentation is ineffective, or equivalent to a solution record.

A second model and a plausible mechanism

On the same primary task, the second model's median turns fell from 31 to 12, counted tokens from 1,187k to 380k, and runtime from 187 to 77 seconds. Every memory run outperformed every baseline run on all three effort measures. Each unadjusted p is 0.0079; treated as a separate family of three, the Holm-adjusted values are 0.024.

FIGURE 4 · EVIDENCE FROM THE EXECUTION TRACES

PRIMARY TASK · BOTH MODELS



Less repeated discovery. The original report identifies seven baseline runs across the two models that executed the removed module-level import and encountered its documented error; none of the ten memory runs did. Calls into installed type definitions also decreased. These transcript summaries are consistent with reduced discovery effort; they do not isolate a causal mediator.

Whole-feature work and the cost of an offer

The whole-feature cohort did not meet its registered benefit criterion. The booking-schema task had modestly lower turns and tokens without a detected difference. The chat-route task had more median turns with memory (12 versus 10), with nearly unchanged runtime; its token counts were confounded by unrelated instructions. Both tasks passed in all runs after the applicable regrading.

The Hono control used more work with memory: 5 versus 4 turns, 171k versus 130k counted tokens, and 65 versus 59 seconds. None of these differences reaches an unadjusted 0.05 threshold. Nevertheless, the direction illustrates a practical cost: a record must be retrieved, read, and assessed even when it adds little.

The two original knowledge controls also prevent a clean distinction between “new” and “known” material. Their median runtime ratios favor memory by 1.59× and 1.07×. The registered contrast with the smallest API-change ratio, 1.64×, is narrow. The study does not separate calendar recency from demonstrated ignorance, task difficulty, or the amount of construction required.

04 Interpretation and limits

A prior solution is most useful when the missing knowledge is concise, applicable, and expensive to rediscover.

This interpretation explains why a focused API task can benefit more than a whole feature. A record can remove a discovery loop while leaving application structure, integration, testing, and task-specific code to be built. The whole-feature cohort tests that boundary; it does not estimate what fraction of typical development work benefits.

Several limitations affect the strength of the evidence. The same team selected the tasks and authored the records and verifiers. Baseline-first ordering confounds treatment with time and cache state. Provider limits split execution into three windows, and some comparisons cross an environment change. Five runs per arm leave substantial uncertainty; non-significance cannot establish equivalence.

Three verifiers were corrected after execution. The Next.js task required replacement runs because its original verifier could not be passed. AI SDK and Hono verdicts were regraded from preserved artifacts after tests were found to require details absent from the prompts. The Hono correction removed a baseline failure. These changes are retained in the study notes and constrain how pass-rate claims should be read.

Record adoption was also difficult to establish: the study's detector flagged 7 of 65 analyzed offers, all through keyword heuristics. An unflagged record may still have influenced a run. The measurements primarily compare offering memory with baseline, rather than estimating the effect of confirmed adoption.

The September 4 exploratory replay experiments reported larger ratios on repeated tasks, but used only two or three runs per arm and relied on partially recovered records. They remain separate from the main evidence. The source report also records no real-use decisions in the first production readout. An automated policy-improvement attempt showed a development gain but a −0.40% change on sealed validation, and was not promoted. Production impact and a learned retrieval advantage remain unestablished.

What the next study should resolve

A stronger evaluation would freeze the corpus and environment, randomize or counterbalance arm order, and select held-out tasks independently of treatment performance. It should include both old-but-unknown and new-but-known APIs, records supplied by independent teams, and a second agent host. Documentation comparisons need contemporaneous baselines and prespecified equivalence margins.

Operational evaluation should measure abstention, wrong-but-plausible records, and end-to-verification time. A randomized production holdout could establish how often reusable knowledge is available, how often it helps, and whether its savings exceed retrieval and verification costs.

05 Implementation, data, and authorship

Pascal's reference implementation, Blaze, provides the agent client and Solution Card schema.[7] These public components are MIT-licensed; the hosted service implementation and card corpus are outside that repository. The study used a local gateway with hybrid keyword and embedding retrieval, using `Xenova/jina-embeddings-v2-small-en`. Hosted Blaze defaults to lexical retrieval. Its optional dense path uses `openai/text-embedding-3-small` through Vercel AI Gateway, with a 1,536-dimensional model and index. The two configurations are different instruments; this study does not establish the hosted configuration's retrieval quality or production benefit.

The public client treats retrieved records as untrusted reference material and calls for applicability checks and local verification. Contributions require separate review and evaluation. These controls support the workflow; they do not prove the correctness of arbitrary advice or establish complete privacy guarantees.

What is built, and what comes next

The hosted implementation records installation-owned lookups, exact offered revisions, explicit outcomes, and quarantined contributions. Reuse requires applicability checks and local verification. New contributions need exact-content review and supported independent evaluation before becoming eligible; public sharing also requires explicit authorization. Verified consolidation retains source lineage, while revocation and erasure checks can withdraw a source and its derived records.

Observed outcomes currently make a bounded adjustment among already-relevant records. They are agent reports, not independent verification. An offline evaluation and experiment controller can compare candidate policies on separate development and sealed validation tasks; it does not automatically promote a production policy. These are foundations for an improvement loop, not evidence that the deployed system improves itself.

A learned reranker, synthetic query generation, and decomposition of a missed query into subproblems remain research directions. A future guide could rank eligible records or choose abstention, while the coding agent performs the task. No neural search policy or learned retrieval advantage is claimed here. Learning must preserve ownership, version compatibility, verification, and erasure boundaries.

Automatic client hooks send no workspace content. An agent explicitly prepares and reviews a short conceptual query before a lookup. If an operator enables hybrid retrieval, that permitted query can be processed by Vercel AI Gateway and the configured embedding provider; this processing must be disclosed. Application telemetry is configured not to record embedding inputs or outputs, which does not establish a provider's retention policy. A bounded dense-retrieval deadline, scoped cache, and lexical fallback limit the impact of provider failure.

Data availability. This edition presents numerical summaries checked against the September 7–8 study rows and the original September 8 report. The authors retain the preregistration, amendments, per-run evidence, harness, and analysis code privately. No public reproducibility supplement accompanies this edition. Figures use the reported medians and ranges; independent readers cannot reconstruct the experiment from this page alone.

Author contributions and interests. Aymeric Rabot and Julien Brissonneau develop the system at Pascal and have a commercial interest in it. They selected the tasks, authored the records and verifiers, and ran and analyzed the study.

06 References

1. Ellis, K., Wong, C., Nye, M., et al. (2020). DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning. arXiv:2006.08381.
2. Borgeaud, S., Mensch, A., Hoffmann, J., et al. (2021). Improving language models by retrieving from trillions of tokens. arXiv:2112.04426.
3. Wang, G., Xie, Y., Jiang, Y., et al. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291.

4. Ouyang, S., Yan, J., Hsu, I.-H., et al. (2025; revised 2026). ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory. ICLR 2026; arXiv:2509.25140. See also Google Research's account.
5. Tang, X., Qin, T., Peng, T., et al. (2025). Agent KB: Leveraging Cross-Domain Experience for Agentic Problem Solving. arXiv:2507.06229.
6. Wang, Z. Z., Mao, J., Fried, D., and Neubig, G. (2024). Agent Workflow Memory. arXiv:2409.07429.
7. Pascal (2026). Agent client, skill, and Solution Card schema. Public reference components. MIT license.

Appendix A — Complete numerical summaries

Each entry compares baseline → memory. All arms contain five runs. "Tokens" means the total counted tokens defined in Section 2. Values are transcribed from the original report, retaining its rounding. Percentages in Figure 2 are calculated from those displayed medians.

Table A1. Main task comparisons: baseline → memory.

Task	Turns, median	Tokens, median	Runtime, median (s)	Passes
Vitest 5 benchmark · cross-repository	19 → 10	893k → 383k	240 → 111	5/5 → 5/5
Vitest 5 sequential · fixture-authored	13 → 4	431k → 121k	102 → 35	5/5 → 5/5
Stripe thin events · fixture-authored	12 → 10	673k → 469k	295 → 142	5/5 → 5/5
Next.js 16.3 boundary · cross-repository, rerun	13 → 11	628k → 485k	295 → 180	0/5 → 4/5
Zod 4 migration · knowledge control	12 → 7	439k → 235k	138 → 87	5/5 → 5/5
Stripe webhook · knowledge control	12 → 11	531k → 382k	133 → 124	5/5 → 5/5
AI SDK 7 chat route · regraded	10 → 12	617k → 608k*	138 → 136	5/5 → 5/5
Zod 4.5 booking schema	7 → 6	253k → 225k	200 → 129	5/5 → 5/5
Hono CORS widget · control, regraded	4 → 5	130k → 171k	59 → 65	5/5 → 5/5

* AI SDK token medians are confounded: three baseline runs and one memory run loaded an unrelated skill of roughly 100,000 characters. Excluding these leaves two baseline runs at a median of 475k tokens and four memory runs at 603k. This exclusion is descriptive and unbalanced.

Table A2. Primary-task arms and second-model comparison.

Arm / model	Turns: median (range)	Tokens: median	Runtime: median (range), s
Baseline / Fable 5.1	19 (11–33)	893k	240 (101–336)

Arm / model	Turns: median (range)	Tokens: median	Runtime: median (range), s
VKT record / Fable 5.1	10 (8–13)	383k	111 (88–164)
Wrong record / Fable 5.1	16 (13–17)	730k	188 (146–334)
Type definitions / Fable 5.1	17 (11–19)	794k	219 (155–308)
Documentation tool† / Fable 5.1	17 (9–20)	721k	163 (86–264)
Official prose† / Fable 5.1	16 (11–22)	657k	164 (95–197)
Baseline / Opus 4.8	31 (27–32)	1,187k	187 (179–293)
VKT record / Opus 4.8	12 (11–15)	380k	77 (70–89)

All A2 arms passed 5/5. † Later execution window with an environment change. The study used Claude Code with first-party model identifiers `claude-fable-5-1` and `claude-opus-4-8`. Replicate numbers are run labels, not controlled sampling seeds. A complete execution configuration is not included in this edition.

Table A3. Unadjusted, two-sided effort comparisons: memory versus baseline.

Task	*p*, turns	*p*, tokens	*p*, runtime
Vitest 5 benchmark	0.0159	0.0159	0.0556
Vitest 5 sequential	0.0079	0.0079	0.0079
Stripe thin events	0.0952	0.0159	0.0079
Next.js 16.3 boundary	0.7302	0.6905	0.0159
Zod 4 migration	0.3413	0.4206	0.5476
Stripe webhook	0.7302	0.4206	0.6905
AI SDK 7 chat route	0.2222	0.8413*	0.8413
Zod 4.5 booking schema	0.5873	0.5476	0.4206
Hono CORS widget	0.2381	0.0556	0.3095

* Confounded token comparison, retained for traceability. None of the twelve tests in the first four rows survives the reported Holm correction. The Next.js pass-rate comparison has unadjusted Fisher $p = 0.048$. No confidence intervals can be reconstructed from the reported medians and ranges alone.

The Stripe thin-event documentation arm, not plotted among the primary-task controls, had medians of 16 turns, 820k tokens, and 250 seconds, with 5/5 passing. The agent never called its documentation server. The source's index lacked the installed version, but whether that explains the lack of calls was not tested.

Appendix B — Design qualifications and correction history

Registration and task selection

The fresh-knowledge hypothesis was registered before its runs. Exact-repeat and known-knowledge hypotheses emerged from the September 4 exploration; exact repeats were not retested in the main study. One control had already shown little effect, so its selection was partly informed by the expected result. Documentation comparisons and the whole-feature cohort were extensions registered before their own execution.

The original protocol's API-change rule required lower median turns and tokens on at least three of four tasks and a combined turn-test p below 0.05. It was met on four of four tasks. The whole-feature rule was not met. The known-knowledge rule was met on a narrow numerical margin; the wrong-record harm hypothesis was not supported. A model's self-report of its cutoff is not an independently verified account of its training data.

Execution windows and replacements

The report counts 125 protocol and extension runs plus 10 Next.js replacements, for 135 completed runs. Ten original Next.js rows are superseded. Seventy-five provider-refused attempts were excluded and the affected arms rerun. Replicate numbers identify repeated runs, not controlled sampling seeds.

Provider limits split the study into three windows. Two dependencies and three corpus records were added between the first two. The three documentation arms were compared with earlier baselines. The booking-schema baseline straddled the second interruption. Next.js replacements ran in the third window. No repeated baseline isolated drift across all windows.

Verifier amendments

Task	Defect	Correction and reporting
Next.js boundary	Incompatible export requirements made the original verifier unpassable, including by the reference solution.	Both arms rerun under a fixed verifier; original rows superseded. Review still identified an overly permissive scope check.
AI SDK chat route	A test required a payload field name absent from the task prompt.	Both arms regraded from preserved artifacts; all ten pass.

Task	Defect	Correction and reporting
Hono CORS widget	A test imposed a particular status code where the prompt allowed more than one reading.	Both arms regraded; a baseline failure is removed and all ten pass.

The Next.js rerun also illustrates instability in effect size. Its original median turn comparison was 20 → 15; approximately five hours later it was 13 → 11. The corresponding token comparisons changed from 930k → 624k to 628k → 485k. The direction persisted, but its magnitude varied.

Inference and measurement

At five runs against five, the smallest attainable exact two-sided permutation p is 0.0079. Fisher's combined test assumes independent tasks; related tasks and shared conditions qualify that assumption. The correction families were chosen after the registered protocol. Reported tests do not establish practical equivalence.

The adoption detector flagged 7 of 65 analyzed offers: five Next.js runs and two booking-schema runs. None was an explicit agent declaration. Fourteen user-level connectors were attached in every run but unused. An unrelated bundled skill affected four AI SDK runs, unequally across arms. One wrong-record run performed the study's only three web fetches.

Exploratory exact replay

September 4 runs repeated the same prompt in the same repository with stored final files. Reported runtime ratios ranged from 3.6–3.8× on the Vitest benchmark, reached 6.1× on a Zod hot path, and ranged from 2.5–4.6× on four everyday builds, with two or three runs per arm. The original per-run file was partly lost; 34 rows were recovered, including one earlier smoke run.

The September 4 public-site recordings are a separate exploratory set. The source report describes ten displayed pairs at approximately 3.2× aggregate agent runtime, with five same-repository repeats. An eleventh recording was omitted from the displayed set; including it would count one baseline twice. These selected recordings do not measure general transfer or production benefit and are excluded from this edition's headline result.

The fuller study report and correction history are retained by the authors. The condensed notes above preserve the qualifications material to the claims made here; the reproducibility materials have not been publicly released.